PUMA2– Short User Guide

Detlev Droege

21.11.2007

Arbeitsgruppe Aktives Sehen (AGAS)
Institut für Computervisualistik
Universität Koblenz-Landau
Universitätsstraße 1, 56070 Koblenz
droege@uni-koblenz.de
http://www.uni-koblenz.de/~droege

Inhaltsverzeichnis

1 Introduction **3**
1.1 Prerequisites 3

2 Creating applications with PUMA2 **3**
2.1 Makefile 3
2.2 Program 4
2.3 Using the grabber library 4

3 Documentation **6**

Document history

Version	State	Date	Author(s)	Description
0.1	in work	21.11.2007	DD	initial version.
0.2	in work	30.11.2007	DD	added sample code.

1 Introduction

This document describes the basic 'usage' of PUMA2, that is the end user perspective to the packages provided e.g. as RPM packages. These contain libraries and header files to develop applications based on PUMA2. The term 'end user' is meant from the perspective of the PUMA2 developer team. In this sense, end users are those developers using the PUMA2-Libraries to develop their own applications.

1.1 Prerequisites

Currently, PUMA2-packages are distibuted as RPM files only. Following packages are available currently:

`libpuma2` This is the core library and must be installed in any case to develop PUMA2 based applications. It provides the basic image classes and a limited number of image operations.

Currently, this package requires the presence of ImageMagick++, OpenCV, libpng and libjpeg.

`libpuma2grabber` This package provides access to live video data. Based on the UniCap[1] library it allows the use of cameras attached via FireWire, USB or specialized capture hardware.

Obviously, all required packages have to be installed using the usual mechanisms (rpm or some high level package management tool like Yast2). The standard development tools expected are the GNU-C++ compiler suite (g++) and the GNU make program.

2 Creating applications with PUMA2

2.1 Makefile

Unless an integrated development environment is used, which normally create makefiles on their own, the usage of makefiles for controlling the code generation is highly recommended.

As PUMA2 provides the necessary information for the increasingly common tool pkg-config, writing makefiles is is quite simple.

```
CXXFLAGS = $(shell pkg-config --cflags libpuma2)
LIBS     = $(shell pkg-config --libs libpuma2)

SRC = minidemo.cc
OBJS = $(SRC:.cc=.o)

demo: $(OBJS)
      $(CXX) -o $@ $(CXXFLAGS) $(OBJS) $(LIBS)
```

(Be sure to check that the indented command lines in the Makfile start with a TAB character, not with a series of blanks!)

[1] http://unicap.sf.net/

2.2 Program

The C++ program your write of course has to include some header files, like image classes and operations. In most cases (for now) you will use `GrayLevelImage8` or `ColorImageRGB8`. An example:

```
1  #include <ColorImageRGB8.h>
2  #include <ImageWriter.h>
3  #include <ImageReader.h>
4
5  using namespace Puma2_NS;
6
7  int main (int argc, char **argv)
8  {
9      ColorImageRGB8      cin(0,0);
10     std::string         cfilename ("some-image.jpg");
11
12     ImageReader::readImage(cin, cfilename);
13
14     int pleft  =   cin.getWidth()/4;
15     int pright = 3*cin.getWidth()/4;
16     int ptop   =   cin.getHeight()/4;
17     int pbottom= 3*cin.getHeight()/4;
18
19     for (int y = ptop; y < pbottom; y++) {
20        cin[y][pleft][0] = 255;
21        cin[y][pleft][1] =   0;
22        cin[y][pleft][2] =   0;
23        cin[y][pright][0] =   0;
24        cin[y][pright][1] = 255;
25        cin[y][pright][2] =   0;
26     }
27     for (int x = pleft; x < pright; x++) {
28        cin[ptop][x][0] = 255;
29        cin[ptop][x][1] =   0;
30        cin[ptop][x][2] = 255;
31        cin[pbottom][x][0] = 255;
32        cin[pbottom][x][1] = 255;
33        cin[pbottom][x][2] =   0;
34     }
35
36     ImageWriter::writeImage (cin, "marked-image.png");
37 }
```

2.3 Using the grabber library

To capture images from live cameras, the PUMA2 extension library **`libpuma2grabber`** is used. After installing this additional package the following code should grab 50 frames from one of the attached cameras (you have to answer interactively which one to use). Store the following C++ source code as **`simpleGrabber.cpp`**.

```
1  /*****************************************************
2   * a simple demo application using the puma2grabber
3   * to read some frames into puma2 color images and write
4   * them to disk
5   *
6   * Author: Benjamin Knopp <bknopp@uni-koblenz.de>
7   *****************************************************/
8  #include "GrabbingDeviceManager.h"
9  #include "GrabbingDevice.h"
10
11 #include <ColorImageRGB8.h>
12 #include <ImageWriter.h>
13
14 using namespace Puma2_NS;
```

```
15  using namespace std;
16
17
18  // grab a maximum of 15 frames per second
19  static const float FPS = 15.0;
20  static const int NUM_FRAMES = 50;
21
22  int main(int argc, char **argv)
23  {
24          // get instance of GrabbingDeviceManager
25          GrabbingDeviceManager* myManager =GrabbingDeviceManager:: getGrabbingDeviceManager();
26
27          // check if supported devices are present
28          if ( ! myManager->getNumDevices())
29          {
30                  cerr << "no supported devices found, sorry" << endl;
31                  return EXIT_FAILURE;
32          }
33
34          // list all supported devices
35          myManager->printDeviceList();
36
37          unsigned int devNum = myManager->getNumDevices() +1 ;
38
39          while ( devNum >= myManager->getNumDevices() )
40          {
41                  cout << "please enter device number" << endl;
42                  cin >> devNum;
43          }
44
45          GrabbingDevice *myCam;
46          // connect to GrabbingDevice
47          bool camFound = myManager->connectGrabbingDevice( &myCam, devNum );
48          if ( !camFound ) return EXIT_FAILURE;
49
50          myCam->printFormatsList();
51
52          // choose first camera format which could be converted to rgb
53          myCam->setFormat2("width=* height=* in=* out=RGB");
54          // start capture process
55          myCam->startCapture();
56
57          // puma2 rgb image
58          ColorImageRGB8 image;
59          string filename;
60
61          // capture NUM_FRAMES frames
62          for( int i = 0; i < NUM_FRAMES; i++ )
63          {
64                  // usleep( (int) (1000000.0 / FPS) ); // should be "minus grabtime"
65                  // try to grab image and write it to disk
66                  if ( myCam->grabImage( image ))
67                  {
68                          cout << "\b\b" << toString(i) << flush;
69                          filename = "frame_" + toString( i ) + ".jpg";
70                          // WARNING: writing png to disk is SLOW!
71                          ImageWriter::writeImage(image, filename.c_str());
72                  }
73          }
74          // stop capturing process
75          myCam->stopCapture();
76          delete myCam;
77
78          return EXIT_SUCCESS;
```

```
79  }
```

For the makefile, with respect to the one shown in section 2.1, it is sufficient to change the pkg-config package parameter to **libpuma2grabber**, as this will automatically include **libpuma2** (and other libraries needed) as well.

```
1   INCPATH = $(shell pkg-config --cflags libpuma2grabber)
2   LIBPATH = $(shell pkg-config --libs   libpuma2grabber)
3
4   CXXFLAGS  = -g -O2 $(INCPATH)
5   LDFLAGS   = -g $(LIBPATH)
6
7   all: simpleGrabber
8
9   simpleGrabber: simpleGrabber.o
10          $(CXX) simpleGrabber.o $(LDFLAGS) $(CXXFLAGS) -o simpleGrabber
11
12  clean:
13          -rm simpleGrabber simpleGrabber.o
```

3 Documentation

... is currently very poor.
Please refer to the doxygen-generated API description at http://www.uni-koblenz.de/puma/doc.